TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: B2646 – Informační technologie
Studijní obor: 1802R007 – Informační technologie

ePCBlab – klientské rozhraní

ePCBlab – client interface

Bakalářská práce

Autor: **Jakub Petržílka**
Vedoucí práce: prof. Ing. Zdeněk Plíva, Ph.D.
Konzultant: Ing. Leoš Petržílka

V Liberci 11. 5. 2011

(Originální zadání práce)

Prohlášení

Byl(a) jsem seznámen(a) s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé bakalářské práce a prohlašuji, že **s o u h l a s í m** s případným užitím mé bakalářské práce (prodej, zapůjčení apod.).

Jsem si vědom(a) toho, že užít své bakalářské práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Bakalářskou práci jsem vypracoval(a) samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce a konzultantem.

Datum

Podpis

Děkuji Martinu Pomeznému, tvůrci serverové části ePCBLabu, za kooperativní přístup při vývoji systému. Dále děkuji ing. Leoši Petržílkovi, vedoucímu PCB Laboratoře, za úvod do problematiky a za ochotu konzultovat každý detail připravovaného systému.

Abstrakt

Tato práce se zabývá realizací dvou klientských uživatelských rozhraní pro evidenční a informační systém ePCBLab. Systém ePCBLab je navržen pro laboratoř na Technické univerzitě v Liberci, zabývající se návrhem a výrobou desek plošných spojů. V první části této práce je popsáno, jakým způsobem byly vybrány technologie pro realizaci systému. Další část se věnuje navržené architektuře systému, popisuje význam jeho jednotlivých částí, komunikační protokol propojující tyto části a navržený datový model. V poslední části jsou popsána obě vytvořená klientská rozhraní a jejich struktura z programátorského i uživatelského hlediska.

Klíčová slova: PCBlab, správa zakázek, klientské rozhraní, evidenční systém, evidence, XOE, JSP, WPF

Abstract

This thesis deals with realizations of two client user interfaces for the registration and information system ePCBLab. This system is designed for laboratory at Technical University of Liberec, which is engaged in design and production of printed circuit boards. At the first part of this text is described how technologies for realization of this system ware selected. Next part describes designed architecture of ePCBLab system and its individual parts, communication protocol connecting those parts together and considered data model. The last part of this text is dedicated to description of both developed client user interfaces and their structures in terms of programmer and user.

Keywords: PCBlab, contract management, client interface, registering system, XOE, JSP, WPF

Obsah

Seznam použitých zkratek

ASCII **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange, základní kódová tabulka znaků

CSS **C**ascading **S**tyle **S**heet, kaskádové styly webových stránek

DPS **D**eska(-y) **P**lošných **S**pojů, ekvivalent anglického **PCB**

EOF **E**nd **O**f **F**ile, konec souboru, speciální znak v ASCII

HTML **H**yper**T**ext **M**eta **L**anguage, jazyk pro popis obsahu webových stránek

HTTP **H**yper**T**ext **T**rasnport **P**rotokol, protokol aplikační vrstvy používaný pro přenos obsahu webových stránek po internetu

JSP **J**ava **S**erver **P**ages, technologie pro vývoj webových stránek

MVC **M**odel **V**iew **C**ontroller, architektonický vzor pro vývoj aplikací

PCB **P**rinted **C**ircuit **B**oard(s), Deska(-y) plošných (tištěných) spojů

SHA-1 hašovací funkce, název je odvozen od **S**ecure **H**ash **A**lgorithm

SOAP **S**imple **O**bject **A**ccess **P**rotocol, protokol určený pro výměnu XML zpráv po síti

SSL **S**ecure **S**ockets **L**ayer, vrstva zprostředkovávající zabezpečený přenos dat sítí

TCP **T**ransmission **C**ontrol **P**rotocol, protokol pro přenos dat v internetu, se zaručeným doručením

UTF-8 znaková sada

WPF **W**indows **P**resentation **F**oundation, součást Microsoft .NET frameworku, která umožňuje tvorbu uživatelského rozhraní

XAML e**X**tensible **A**plication **M**arkup **L**anguage, rozšíření XML, které se používá pro popis uživatelského rozhraní WPF aplikací

XML e**X**tensible **M**arkup **L**anguage, obecný rozšířitelný značkovací jazyk

XSD **X**ML **S**chema **D**efinition, šabolny pro definici obsahu XML souborů a zpráv

Seznam obrázků

Seznam tabulek

Konvence v sazbě

- *Kurzívou* s počátečním velkým písmenem jsou psány názvy tříd, entit, atributů, XML elementů, souborů, jmenných prostorů a balíčků.
- ***Tučnou kurzívou*** jsou psány názvy tlačítek, záložek, aplikačních oken, formulářů a ostatních prvků uživatelského rozhraní
- `Neproporcionálním` písmem jsou psány ukázky kódu a obsah souborů
- „V uvozovkách“ jsou psány neoficiální, zažité nebo slangové pojmy
- S podtržením jsou psány odkazy na webové stránky

1 Úvod

Problematika správy, evidence a řízení zakázek ve firmách a organizacích je v poslední době velmi populárním tématem. Ve snaze zefektivnit výrobu je řada z nich nucena, kromě zavádění automatizace do výroby, automatizovat a usnadňovat i veškerou agendu související s výrobou. Přestože dnes existuje celá řada univerzálních systémů pro správu zakázek, tendence jsou takové, že se každá expandující organizace snaží vytvořit vlastní systém, uzpůsobený přesně podle vlastních požadavků. Realizace každé jedné zakázky může být v některých případech velmi komplexní záležitostí a může záviset na řadě parametrů. Aby mělo vedení organizace detailní přehled o výrobě a nemuselo archivovat podklady pro realizaci jednotlivých zakázek zvlášť, je nutné tyto parametry zaznamenávat centrálně v evidenčním systému.

Výše popisovanou situaci je možné pozorovat i v univerzitní laboratoří PCBLab na Technické univerzitě v Liberci. Tato laboratoř se již řadu let zabývá návrhem a výrobou desek plošných (tištěných) spojů. Zkratka PCB pochází z anglického pojmu printed circuit boards. Výrobní proces desek plošných spojů je velmi složitý a skládá se z mnoha výrobních operací. Některé z nich jsou parametrizované, jiné jsou zcela volitelné. U každé zakázky má zákazník možnost zvolit jednotlivé požadované operace a jejich parametry. Tyto parametry zakázky je tedy nutné zaznamenávat do evidenčního systému.

Dobrý evidenční systém zakázek by však měl ve své funkčnosti jít mnohem dál. Je potřeba automaticky na základě různých finančních, spotřebních, mzdových a jiných konstant automaticky vypočítávat cenu zakázky, množství spotřebovaného materiálu, mzdy pro jednotlivé zaměstnance a generovat různé přehledové statistiky. Nakonec je také vhodné automaticky generovat všechny potřebné zakázkové dokumenty, jako je průvodka, předávací protokol, podklady pro účetnictví a umožňit jejich tisk.

Tato práce se zabývá realizací dvou klientských uživatelských rozhraní evidenčního a informačního zakázkového systému ePCBLab, určeného pro univerzitní laboratoř PCBLab. Vzhledem k vybranému způsobu realizace je v této práci věnována velká pozornost jádru systému, kterým se stal komunikační protokol, propojující zmíněná klientská uživatelská rozhraní s aplikačním serverem ePCBLabu. Tento protokol jsme navrhli ve spolupráci s Martinem Pomezným, který je autorem aplikačního serveru.

2 Volba způsobu realizace

Způsob realizace vyplynul z rozsáhlé analýzy, kterou jsme připravili s Martinem Pomezným v bakalářském projektu v minulém roce. V rámci této softwarové analýzy jsme nastudovali veškeré podklady, které nám byly poskytnuty. Především se jednalo o dosavadní evidenční dokumenty vedené v podobě sešitů aplikace Microsoft Excel. Každá zakázka představovala jeden samostatný sešit s mnoha listy, nesoucí sebou všechny výpočetní vzorce. Na základě těchto podkladů byl v projektu zkonstruován předběžný datový model připravovaného evidenčního systému a předloženy tři možné způsoby technické realizace evidenčního systému:

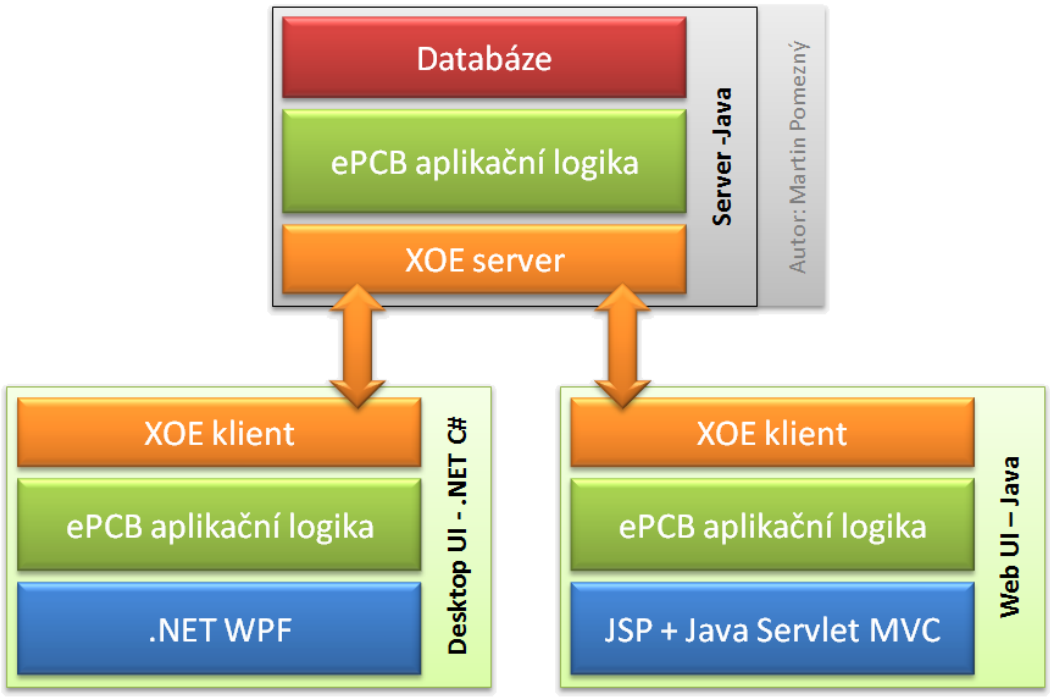
- Jediná webová aplikace, ve které by bylo stejné uživatelské rozhraní pro zaměstnance i pro zákazníky. Zaměstnanci by po přihlášení do systému oproti zákazníkům viděli detailnější obsah a měli by k dispozici více ovládacích prvků.
- Webová aplikace obsahující databázi a rozhraní pro zákazníky, propojená prostřednictvím Web Services s komplexnější desktopovou aplikací určenou zaměstnancům.
- Plně přizpůsobený systém typu klient/server vlastním komunikačním protokolem, obsahující aplikační server s databází, samostatné webové rozhraní pro zákazníky a samostatné klientské desktopové rozhraní pro zaměstnance.

Zatímco v závěru projektu jsme prosazovali zejména první metodu, další analýza nás přivedla k poslednímu zmíněnému způsobu realizace. Tento způsob nám umožnil striktně vymezit rozhraní mezi serverovou a klientskou částí a každý z nás se nadále mohl zabývat pouze svou částí. Realizovat plně přizpůsobený systém typu klient/server sice znamenalo poněkud komplikovanější a pracnější přípravu jádra systému, za to však umožnilo snadnější implementaci uživatelských rozhraní.

Důležitou výhodou vybraného způsobu realizace je také možnost snadné implementace upozorňování zaměstnanců a pověřených pracovníků na změny v systému. Pokud je například zákazníkem vložena přes webové rozhraní do systému nová poptávka a některý ze zaměstnanců má současně otevřené desktopové rozhraní systému, na novou poptávku je ihned upozorněn.

3 Architektura navrženého systému

Systém ePCBLab se skládá ze tří částí. První částí je aplikační server, který vytvořil v rámci své bakalářské práce Martin Pomezný. Server obsahuje databázi, aplikační logiku a komunikační rozhraní. Podrobný popis serveru a jeho funkčnosti však není předmětem této práce. Další částí je klientská desktopová aplikace, která slouží jako uživatelské rozhraní pro zaměstnance laboratoře nebo další pověřené pracovníky. Poslední část je pak tvořena klientským webovým rozhraním, které je určené zákazníkům. Tyto tři části pojí společný komunikační protokol, který byl pro tento typ systému speciálně navržen. Nazvali jsme jej XOE (XML Object Exchange) a jeho detailní popis se nachází v kapitole 3.1. Popis obou klientských částí aplikace naleznete v kapitolách 4 a 5. Architektura celého systému je znázorněna na obrázku (Obrázek 3.1)dole.



Obrázek 3.1 Navržená architektura systému ePCBLab

Kromě XOE lze za další pojítko mezi zmíněnými třemi částmi systému považovat navržený datový model, který bylo nutné implementovat zvlášť a odpovídajícím způsobem pro každou ze tří částí systému. Datový model je svázán s vrstvou aplikační logiky a na předchozím obrázku (Obrázek 3.1) není samostatně zakreslen. Jako prostředek pro implementaci datového modelu v klientské desktopové aplikaci byl použit diagram tříd znázorněný v příloze (Příloha A – diagram tříd

datového modelu). Diagram byl vytvořen v prostředí Microsoft Visual Studia 2010. Podrobně je datový model popsán v kapitole 3.2.

Systém ePCBLab umožňuje díky své architektuře současnou interakci libovolného počtu pracovníků podílejících se na chodu laboratoře a libovolného počtu zákazníků využívajících jejich služeb. Přičemž nemůže nastat žádný konflikt. Editace položek je zabezpečena důmyslným systémem zámků, který znemožňuje více uživatelům editovat stejné položky ve stejný okamžik. Tento systém zámků je podrobněji popsán v kapitole 3.1, pojednávající o komunikačním protokolu XOE. Úkony provedené z webového rozhraní systém okamžitě propaguje ostatním právě připojeným aplikačním klientům. Pracovníci laboratoře tak mohou operativně spolupracovat a reagovat na požadavky zákazníků.

Navržená architektura je připravena k dalšímu rozšiřování funkčnosti. Komunikační vrstva je striktně oddělena od aplikační logiky, proto není problém přidávat například další položky do evidence, modifikovat uživatelská rozhraní a podobně. Faktem je, že změna musí být vždy provedena jak na serveru, tak obou typů u klientů.

3.1 Komunikační protokol XOE

Protokol XOE (XML Object Exchange) slouží jako komunikační nástroj mezi všemi částmi systému ePCBLab. Zajišťuje přenos dat mezi serverem a připojenými klienty pomocí XML zpráv. Byl navržen jako alternativa k protokolu SOAP, který se často využívá v rámci Web Services nad protokolem HTTP. Protokol XOE je, na rozdíl od zmíněné alternativy, stavový protokol. Spojení je možné udržovat persistentně navázané. Tato vlastnost umožňuje snadné a okamžité informování všech účastníků připojených do systému o všech událostech, které nastanou. Dále XOE řeší přístup více uživatelů ke stejné datové položce ve stejný okamžik a definuje systém zámků položek otevřených pro zápis. Protokol XOE nevyužívá žádný jiný protokol aplikační vrstvy, definuje vlastní formát přenášených zpráv. Přenos je realizován výhradně přes TCP spojení zabezpečené vrstvou SSL. Server obsahuje certifikát podepsaný certifikační autoritou. Pokud certifikační autorita není důvěryhodná, je nutné certifikát importovat do úložiště na straně klienta, aby bylo možné ověřit identitu serveru.

XOE protokol je univerzální a jeho implementace jsou v samostatných knihovnách, je tedy možné jej použít i pro realizaci jiné síťové aplikace, kde se

přenášejí komplexnější datové struktury. Příkladem takové aplikace by mohla být i online hra, která neklade příliš velké nároky na rychlost odezvy. XOE není vhodné použít v aplikacích náročných na vysoký datový tok. XML formát má relativně velkou režii, především při zapouzdřování binárních dat.

3.1.1 Druhy přenášených zpráv

V rámci XOE jsou definovány tři druhy zpráv. Struktury těchto zpráv jsou popsány XSD šablonami, které musí být v jednotlivých implementacích tohoto protokolu obsaženy. Každá přijatá zpráva je podrobena validaci proti příslušné XSD šabloně. Prvním typem zprávy je požadavek (*Request*), který je zasílán vždy od klienta na server. *Request* obsahuje vždy svůj identifikátor unikátní v rámci relace mezi serverem a klientem (*Id*) a dále typ požadavku (*Type*). Přehled jednotlivých typů je v tabulce (Tabulka 1). Volitelně mohou být připojena data související s požadavkem (*Data*), v případě autentifikačního požadavku i uživatelská identita (*Identity*) a příznak, jestli klient chce být upozorňován serverem na události (*Notify*). Autentizace je popsána v kapitole 3.1.3. Identita obsahuje uživatelské jméno (*Username*) a otisk hesla ve formátu SHA1 (*PasswordHash*). Dalším typem zprávy je odpověď (Response), která je naopak posílána vždy ze serveru ke klientovi. *Response* opět obsahuje identifikátor (*Id*), pomocí kterého se bezprostředně se váže k předcházejícímu požadavku. Kromě identifikátoru je v odpovědi také stavový kód (*Status*), který klienta informuje o tom, jak dopadl jeho požadavek. Přehled stavových kódů je v tabulce (Tabulka 2). Volitelně mohou být opět připojena související data a speciálně i serverem požadované parametry komunikace (*Hello*). Tyto parametry obsahují informace o verzi serveru (*Version*) a o požadované minimální četnosti informování o přetrvávání spojení (*Heartbeat*). Parametry komunikace se posílají v odpovědi na autorizační požadavek. Posledním typem zprávy je upozornění (*Notification*), které se posílá ze serveru ke všem připojeným klientům, kteří si požádali o doručování těchto notifikací. Upozorňovací zpráva obsahuje typ upozornění (*Type*) a související data (*Data*). Přehled a popis typů upozornění je v tabulce (Tabulka 3).

Tabulka 1 Druhy požadavků definovaných v XOE protokolu

Typ požadavku	Význam
Get	Načíst kolekci data ze serveru na základě vzoru
Put	Uložit datovou položku na server
Update	Aktualizovat datovou položku na serveru
Delete	Smazat datovou položku na serveru
Lock	Uzamknout editaci datové položky na serveru
Auth	Autorizovat se na serveru pomocí přiložené identity
Heartbeat	Upozornit server o přetrvávajícím spojení

Tabulka 2 Stavy odpovědí XOE protokolu

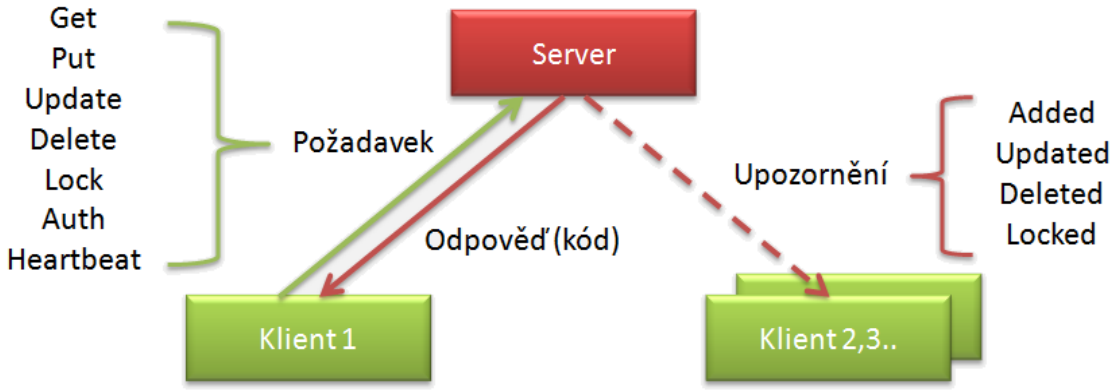
Stavový	Název stavu	Význam
100	HEARTBEAT_OK	Potvrzení oznámení o přetrvávání spojení
101	AUTH_OK	Autentizace proběhla úspěšně
200	QUERY_OK	Požadovaná operace s daty proběhla úspěšně
210	LOCK_OK	Položka byla úspěšně uzamčena
300	BAD_REQUEST	Nepovolený formát dotazu
301	AUTH_FAILED	Autentizace selhala
401	QUERY_BAD_DATA	Požadovaná operace selhala, nepřijatelná data (pro Put nebo Update)
402	QUERY_FAILED_LOCKED	Požadovaná operace selhala, položka je uzamčena (pro Update nebo Delete)
403	QUERY_NOT_PERMITTED	Požadovaná operace selhala, nedostatečná oprávnění
404	QUERY_DATA_NOT_FOUND	Požadovaná operace selhala, Data nenalezeny (pro Get)
413	LOCK_FAILED_LOCKED	Uzamčení selhalo, položka je již uzamčena
414	LOCK_FAILED_NOT_FOUND	Uzamčení selhalo, položka neexistuje

Tabulka 3 Druhy upozornění definované v XOE protokolu

Typ upozornění	Význam
Added	Do systému byla přidána nová datová položka
Updated	V systému byla aktualizována datová položka
Deleted	Datová položka byla smazána ze systému
Locked	Datová položka v systému byla uzamčena

3.1.2 Princip komunikace

Komunikace probíhá podle schématu definovaného čtyřmi kroky. V prvním kroku klient sestaví požadavek (*Request*) a odešle na server. V dalším kroku server požadavek vyhodnotí. Pokud je možné požadavku vyhovět, provede příslušnou změnu v datech. Ve třetím kroku server připraví odpověď (*Response*) a odešle ji klientovi. V posledním kroku server zkontroluje, jestli byla data požadavkem ovlivněna. V případě, že ano, rozešle upozornění (*Notification*) ostatním klientům, kteří požádali o doručování těchto notifikací. Princip fungování protokolu a předávání jednotlivých druhů zpráv je naznačen na následujícím obrázku (Obrázek 3.2).



Obrázek 3.2 Princip komunikace XOE protokolu

3.1.3 Autentizace

Autentizaci (tedy ověření uživatele) je nezbytné provést ihned po navázání spojení. Klient pošle na server požadavek na autentizaci (*Request* typu *Auth*) a přiloží uživatelskou identitu (*Identity*) a parametr určující jestli má být upozorňován na události v systému (*Notify*). Pokud se nepošle autentizační požadavek jako první nebo ověření uživatele a hesla selže, server odpoví příslušným stavovým kódem a uzavře spojení. Pokud je autentizace úspěšná, odpoví klientovi příslušným stavovým kódem a k odpovědi připojí úvodní informace o parametrech navázaného spojení (*Hello*).

3.1.4 Udržování spojení

Na serveru je nastaven časový limit, který je na začátku komunikace klientům oznamován v parametrech spojení. Minimálně jednou v tomto časovém limitu se musí jednotliví připojení klienti serveru ozvat. Pro tento účel existuje v XOE typ požadavku *Heartbeat*. Pokud se neozvou, jsou serverem považovány za neaktivní a spojení s nimi je ze strany serveru ukončeno.

3.1.5 Oddělování jednotlivých zpráv

Na straně klienta i na straně serveru se příchozí z datového proudu TCP socketu střádají do bufferu. Jednotlivé XOE zprávy jsou od sebe odděleny znakem EOF (čtvrtý znak v ASCII). Tento znak není v XML podle [2] povolen, proto je bezpečným ukončovacím znakem. Pokud přijatá data obsahují tento znak, spojí se s daty v bufferu a vytvoří se z nich příchozí XOE zpráva.

3.1.6 Zamykání položek

Pokud uživatel chce aktualizovat data na serveru, není žádoucí, aby ve stejný okamžik začal data upravovat někdo jiný, protože by zákonitě jeden z uživatelů o své změny přišel. XOE protokol je navržený tak, aby bylo možné zamykat položky, které jsou v daný okamžik editovány. Klientská aplikace nejprve musí na serveru pomocí požadavku na uzamčení položky (*Request* typu *Lock*) zjistit, jestli server umožnil uživateli vybranou položku editovat. Pokud ano, server ji uzamkne a dalšímu uživateli to již nepovolí. Položka se odemkne až poté co uživatel editaci dokončí. Editace může být dokončena buď zasláním požadavku na aktualizaci dat na serveru (*Request* typu *Update*) nebo zahozením provedených změn a znovunačtení položky (*Request* typu *Get*).

3.1.7 Implementace protokolu

V současné době jsou vytvořeny tři implementace XOE protokolu. První z nich je XOE server napsaný v jazyce Java, který je využit v aplikačním serveru ePCBLabu. Další implementací je XOE klient pro platformu .NET, napsaný v jazyce C#, který se využívá v klientské desktopové aplikaci. Poslední implementací je odlehčená verze XOE klienta, která je opět napsána v jazyce Java a je využita v rámci klientského webového rozhraní.

XOE protokol z hlediska jeho implementace definuje striktně pouze formát a strukturu přenášených správ, chování serveru a klienta, požadavky na zabezpečení a průběh komunikace. Struktury pro uchovávání stavů, zámků a obslužné logiky v protokolu blíže specifikovány nejsou. Způsob jejich implementace je libovolný.

3.2 Datový model

Datový model definuje strukturu a formát dat evidovaných v systému ePCBLab. Byl vytvořen na základě požadavků na rozsah a charakter evidovaných informací. Obsahuje pět hlavních entit (*zakázka*, *zákazník*, *materiál*, *operace* a *konstanta*) a několik dalších doplňkových a vazebních entit. Všechny tyto entity jsou spolu více či méně provázané, každá z nich má specifikované konkrétní atributy a závislosti (vazby). V této kapitole je struktura datového modelu blíže objasněna.

Každá entita může obsahovat tři typy atributů. Elementární atributy s jednoduchou hodnotou, které jsou dále označovány jako jednoduché atributy. Atributy založené na jiných entitách, které jsou označovány jako referenční atributy. Tyto atributy definují vazby mezi entitami a mohou být přímé nebo vícenásobné. Přímé obsahují jedinou referenci na konkrétní jednu položku. Vícenásobné obsahují kolekci referencí na libovolný počet položek dané entity. A nakonec atributy reflektující hodnoty z jiných atributů, které jsou označovány jako virtuální atributy.

Všechny entity modelu obsahují atribut *id*, jde o tzv. primární klíč, který slouží k jednoznačné identifikaci položek. V každé hlavní entitě se nachází atribut *pattern*. Tento atribut (někdy též označovaný jako Criterion) slouží pro vyhledání všech položek daného typu, které vyhovují restrikcím v něm uvedeným. Atribut *pattern* odkazuje na příslušnou entitu *pattern*, která definuje všechny atributy, podle kterých je potřeba danou položku vyhledávat v systému.

Centrální entitou evidenčního systému je entita *zakázka*. Je nejobsáhlejší položkou datového modelu a obsahuje největší počet atributů a vazeb. Mezi jednoduché atributy zakázky patří *číslo zakázky*, *název desky*, *dodací lhůta* v jednotkách dní, *počet desek*, *popis*, *přístupový hash* (kód pro online přístup), *zákazníkovo označení* (zákazníkovo číslo objednávky), příznak zdali se jedná o nekomerční zakázku a příznak zdali se jedná o interní zakázku. Do přímých referenčních atributů zakázky se kromě zmíněného patternu řadí také atributy *zákazník* (obsahující odkaz na zákazníka, kterému zakázka patří), *rozměr desky* a *použitý základní materiál* (cílová entita obsahuje cenu, množství a použitý typ materiálu). Vícenásobnými referenčními atributy zakázky jsou kolekce použitého materiálu, požadovaných operací, provedených operací, použitého ostatního materiálu, zvláštních požadavků a kolekce pojmenovaná průběh zakázky. V průběhu zakázky jsou uchovávány reference na položky entity *zakázková událost*,

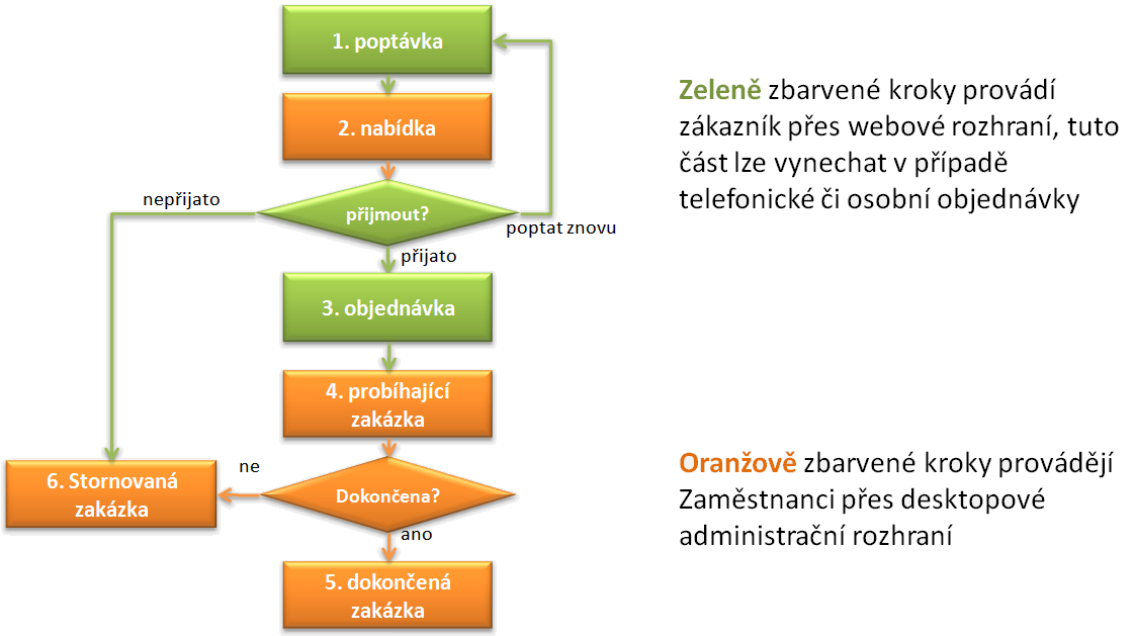
která obsahuje datum a nastávající stav. Z kolekce zakázkových událostí je tedy možné zpětně pozorovat vývoj stavu zakázky. Kolekce provedených operací musí po dokončení zakázky obsahovat stejné reference jako kolekce požadovaných operací. Tímto je zajištěno sledování průběhu výrobních operací na dané zakázce. Virtuálními atributy zakázky jsou *celková cena*, *cena jedné desky*, *současný stav zakázky*, *datum poslední úpravy* a *datum vytvoření zakázky*. Například *současný stav zakázky* reflektuje hodnotu atributu *stav*, obsaženého v poslední zakázkové události (události s nejposlednějším datem) v kolekci *průběh zakázky*. Virtuální atribut *celková cena* reflektuje hodnotu součtu cen provedených operací a použitého materiálu.

Ostatní entity modelu mají strukturu obdobnou. V příloze (Příloha A – diagram tříd datového modelu) je pro ilustraci znázorněný diagram tříd datového modelu, vytvořený v prostředí Microsoft Visual Studia 2010, znázorňující implementaci datového modelu v klientské desktopové aplikaci.

3.3 Životní cyklus zakázky uvnitř systému

Každá zakázka se během svého životního cyklu může nacházet v šesti různých stavech. Postupně budou tyto stavy blíže vysvětleny. Pokud zákazník odešle přes webové rozhraní systému poptávku, vytvoří se v systému nová zakázka se stavem ***poptávka***. Zároveň se vygeneruje přístupový kód, který se zákazníkovi na webu zobrazí a také odešle na uvedený kontaktní e-mail. Ve stejný okamžik se takto nově vzniklá zakázka zobrazí pracovníkům laboratoře v administračním rozhraní desktopového klienta ePCBLabu. Určený pracovník se této zakázky ujme a zkontroluje výrobní požadavky a proveditelnost požadavku. Může upravit parametry a vystavit nabídku. Jakmile takto učiní, systém změní stav zakázky na stav ***nabídka*** a znovu e-mailem upozorní zákazníka, který si opět přes webové rozhraní může nabídku prohlédnout. Zákazník může v tento okamžik učinit rozhodnutí. Nabídku lze přijmout, odmítnout nebo poptat výrobu znovu s jinými parametry. Zakázka takto může projít libovolněkrát fází poptávky a nabídky, dokud zákazník nabídku nepřijme nebo neodmítne. V případě, že zákazník nabídku odmítl, zakázka se ocitne ve stavu ***stornovaná***. Jestliže je nabídka přijata, zakázka přejde do stavu ***objednávka***. V tomto stavu je opět na některém z pracovníků laboratoře, aby objednávku naposledy zkontroloval a přijal k výrobě. Jakmile je zakázka přijata k výrobě, je jí přidělen stav ***probíhající***. Během výroby mohou pracovníci aktualizovat v systému průběh výrobních operací zaškrtáváním již

provedených úkonů. Zároveň mohou zakázku kdykoliv stornovat. Zákazníci mohou během výroby stále sledovat průběh prací na zakázce na webovém rozhraní. Pokud jsou všechny požadované výrobní úkony dokončeny, ePCBLab nastaví stav zakázky na stav ***dokončena***. Pracovníci laboratoře mohou nyní vystavit (vytisknout či elektronicky publikovat) potřebné dokumenty pro předání zakázky. Životní cyklus zakázky je zobrazen na obrázku (Obrázek 3.3 Životní cyklus zakázky).



Obrázek 3.3 Životní cyklus zakázky

4 Desktopové klientské rozhraní

Desktopové rozhraní ePCBLabu je určeno pracovníkům laboratoře. Realizováno je formou klientské aplikace, vytvořené na platformě Microsoft .NET Framework verze 4.0. Zdrojové kódy této aplikace jsou napsané v jazyce C# a značkovacím jazyce XAML, který se stal standardním nástrojem pro tvorbu aplikací využívajících technologii WPF (Windows Presentation Foudation). Pomocí technologie WPF byla vytvořena celá vizuální část desktopového rozhraní.

4.1 Architektura aplikace a struktura zdrojových kódů

Zdrojové kódy jsou rozděleny do šesti ucelených částí, některé z nich jsou univerzální a je možné zkompilovat je i jako samostatné dynamické knihovny. V této kapitole budou všechny tyto části popsány podrobně. Architektura celé desktopové aplikace je zobrazena na obrázku (Obrázek 4.1).

4.1.1 Networking

Knihovna *Networking* zapouzdřuje vytvoření zabezpečeného spojení a komunikaci se serverem na úrovni surových binárních dat. Využívá k tomu TCP socket a komunikační kanál zabezpečený pomocí SSL protokolu. O protokolu TCP a SSL je podrobněji psáno v [1]. Jedná se o univerzální knihovnu, kterou je možné použít i v jiných aplikacích. Je však určena výhradně pro aplikace založené na technologii WPF. Do konstruktoru hlavní třídy této knihovny (*NetworkClient*) se předává reference na objekt, založený na typu *DispatcherObject*. Tento typ je definovaný právě v knihovně WPF a to ve jmenném prostoru *System.Windows.Threading*. Všechny operace prováděné v rámci této knihovny jsou neblokující a provádějí se asynchronním způsobem ve více vláknech. Pokud jsou dokončeny, pomocí metody *Invoke* volané nad zmíněnou referencí na *DispatcherObject* se vyvolá příslušná událost. Tento mechanizmus umožňuje propagovat události ke komponentám uživatelského rozhraní, tím že se jejich obsluha spouští ve vlákně uživatelského rozhraní. Není tedy nutné používat sdílenou paměť a ruční synchronizaci vláken.

Jsou zde definovány dva důležité parametry. Prvním z nich je časový limit pro navázání spojení (*ConnectionTimeout*), který má výchozí hodnotu 10s. Druhým parametrem je četnost kontrol (*PollingInterval*) s výchozí hodnotou 50ms. Oba tyto parametry se dají libovolně nastavit.

Pro příjem dat využívá *NetworkClient* samostatné vlákno, ve kterém se periodicky s četností odpovídající hodnotě atributu (vlastnosti) *PollingInterval* kontroluje buffer příchozích dat. Pokud se v bufferu objeví data, vyvolá se událost, která předá přijatá data vyšší vrstvě. Odesílání dat se provádí jednoduchým voláním metody, která se provádí v hlavním vlákně.

4.1.2 XOE

Knihovna *XOE* implementuje klientské rozhraní protokolu XOE. Nejdůležitější třídou této knihovny je *XoeClient*. Tato třída prostřednictvím dědičnosti rozšiřuje třídu *NetworkClient* a přidává do ní řadu funkcí, atributů a událostí. Kromě surových dat již podporuje navíc příjem a odesílání XOE požadavků, odpovědí a notifikací. Zajišťuje také jejich serializaci do XML, deserializaci z XML a validaci. Přidán je také buffer příchozích zpráv a fronta požadavků. Buffer příchozích zpráv zajišťuje načtení vždy celé jedné zprávy najednou. Fronta požadavků slouží k párování požadavků s odpověďmi. Jakmile je odeslán požadavek na server, je zároveň zařazen do fronty. Odebrán je až v okamžiku, kdy přijde na tento požadavek odpověď. V případě že se během vyřizování požadavku přeruší spojení, je možné opakovat odeslání požadavku z fronty. XOE klient také zajišťuje autentizaci a udržování spojení. Udržování spojení probíhá v samostatném vlákně pojmenovaném *HeartbeatThread*, které periodicky zasílá požadavek typu *Heartbeat* na server. Všechny události ve třídě XoeClient se opět vyvolávají pomocí metody *Invoke* nad *DispatcherObjectem*. Dále jsou v knihovně XOE třídy, které reprezentují konkrétní typy zpráv a položek obsažených uvnitř těchto zpráv. Tyto třídy jsou doplněny o anotace *XMLElement*, které zajišťují pomocí XML serializeru snadný převod všech typů zpráv do XML formátu a zpět. Použitý XML serializer je obsažen v .NET frameworku ve jmenném prostoru *System.Xml.Serialization*. V případě příchozí zprávy *XoeClient* determinuje její typ a ověří ji proti příslušné XSD šabloně. O XSD šablonách pojednává publikace [1]. Všechny XSD šablony, potřebné pro funkčnost XOE protokolu, jsou obsaženy v knihovně XOE. V jednotlivých typech zpráv se vyskytují i nepovinné položky, které je možné při serializaci vynechat. Knihovnu *XOE* lze stejně jako knihovnu *Networking* použít i v jiných aplikacích. Je navržena univerzálně a nezávisle na aplikační logice ePCBLabu.

4.1.3 EPCB

Jmenný prostor EPCB již nemá smysl vkládat do samostatné knihovny, Je vytvořen jen a pouze pro tuto konkrétní aplikaci. Je zde obsažena veškerá aplikační logika, objektové kontejnery uchovávající jednotlivé položky evidence a metody pro manipulaci s nimi. Ve jmenném prostoru EPCB je obsažen *EPCBLabClient*, který představuje nejvyšší úroveň zapouzdření síťové komunikace. Využívá služeb XOE knihovny, ale na venek není vůbec nutné dále s jejími metodami pracovat. Prostřednictvím třídy *EPCBLabClient* je možné na serveru manipulovat přímo s položkami konkrétního typu. Například se zakázkou. Všechny třídy v tomto jmenném prostoru jsou závislé na třídách ze jmenného prostoru *DataModel*. Ve třídě *EPCBLabClient* je použit pro serializaci jednotlivých položek stejný XML serializer, jako v *XOE* knihovně.

4.1.4 DataModel

Dalším jmenným prostorem je *DataModel*, který implementuje navržený datový model. Implementace je zajištěna sadou tříd. Každá z nich představuje jednu entitu. V *DataModelu* je tedy definováno, jakým způsobem budou evidovaná data reprezentována v paměti programu. Oproti obecné definici datového modelu ePCBLabu je zde navíc u pěti hlavních entit jednoduchý atribut *IsLocked*, který indikuje, zdali je daná položka uzamčena. Dále jsou zde zahrnuty anotace *XMLElement*, které určují, jakým způsobem mají být jednotlivé typy položek serializovány do XML a jakým způsobem mají být zpětně deserializovány. V některých typech položek jsou zahrnuty také atributy, které mají pomocný význam a do XML se nepřevádějí. V takovém případě jsou označeny anotací *XMLIgnore*. Všechny atributy jednotlivých položek jsou z hlediska instance volitelné a nemusejí mít nastavenou žádnou hodnotu. Pokud jsou referenčního typu, neserializují se v případě, že nemají přiřazenou konkrétní hodnotu (hodnota je *null*). Význam hodnoty *null* je vysvětlen v [3]. V případě, že se jedná o elementární datový typ, je použit příslušný typ implementující rozhraní *INullable*. Také je společně s daným atributem definována metoda *ShouldSerialize*, která kontroluje přítomnost konkrétní hodnoty různé od *null* a určuje tak, zda má být atribut serializován a odeslán ke zpracování na server.

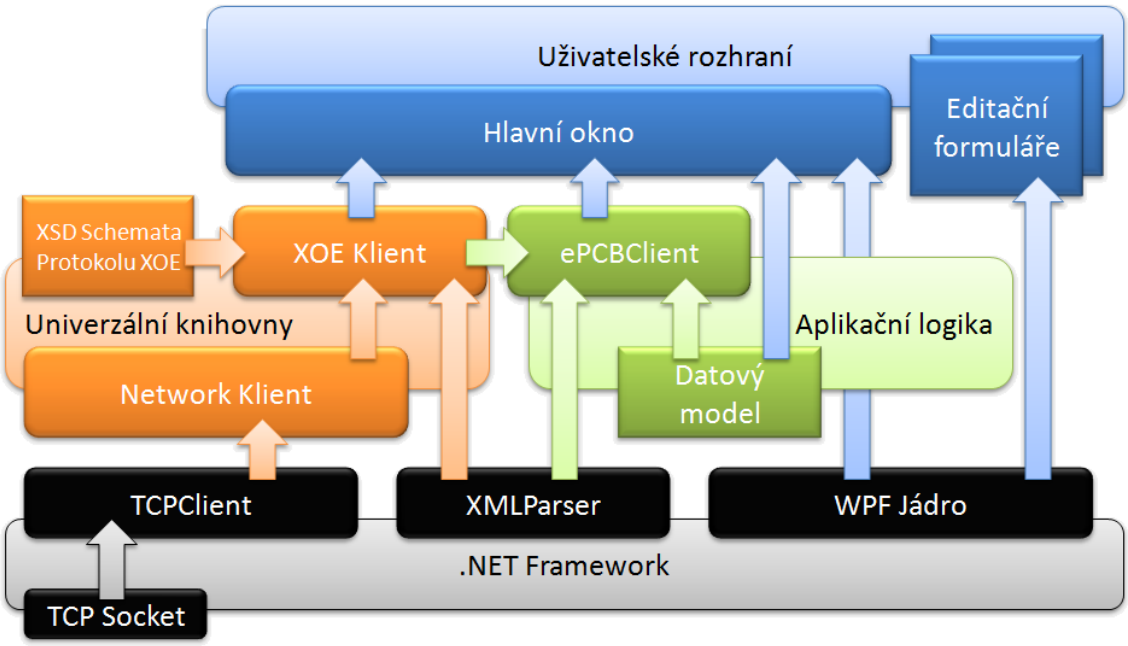
4.1.5 Functions

Pro drobné univerzální funkce různého charakteru je vytvořena samostatná knihovna *Functions*. V této knihovně jsou třídy a metody pro práci se surovými daty, pro klonování objektů, UTF-8 kódování řetězců, generování SHA-1 otisku a zaznamenávání událostí do logu.

4.1.6 UserInterface

Uživatelské rozhraní je také oddělené v samostatném jmenném prostoru pojmenovaném *UserInterface*. Obsahuje vstupní metodu programu. Reference na instance všech knihovních tříd jsou uchovávány přímo v jednotlivých oknech a formulářích vizuálního rozhraní. Každá třída v *UserInterface* (vyjma třídy *App*) představuje jedno okno nebo jeden formulář a je rozdělena vždy do dvou samostatných souborů. V prvním z nich je deklarativní část, popsaná značkovacím jazykem XAML. Význam využití jazyka XAML je popsán v publikaci [5]. Ve druhém souboru je pak část s funkční logikou, která je napsána v jazyce C#. Třída App obsahuje vstupní metodu programu a zajišťuje inicializaci programu. Tato struktura zdrojových souborů uživatelského rozhraní vychází z použití vývojového prostředí Microsoft Visual Studio 2010.

Zajímavou technikou, kterou nabízí technologie WPF je stylování vizuálních komponent. Tato technika se velice podobá stylování webových stránek pomocí kaskádových stylů (CSS). V této aplikaci je stylování použito na každém formuláři. Zajišťuje přehlednější a úspornější kód a možnost styly snadno a rychle obměňovat. Pomocí stylů je zajištěno například to, že všechny tabulky v aplikaci používají stejné barevné schéma, velikosti řádků apod. Další použitou technikou je Data Binding, který je ve WPF velmi propracovaný. Přímo ve značkovacím jazyce XAML lze s jednotlivými prvky na formuláři svázat konkrétní položku v dané třídě. Za běhu aplikace se pak již pouze dynamicky přiřadí konkrétní instance do datového kontextu formuláře. Za zmínku stojí také použití WPF Ribbon knihovny, která obsahuje komponenty, ze kterých je možné vytvořit ribbonové menu. Tento typ menu je známý například z aplikací z novějších verzí sady Microsoft Office.



Obrázek 4.1 Struktura klientské desktopové aplikace

4.2 Struktura a popis uživatelského rozhraní

Uživatelské rozhraní desktopové aplikace je tvořeno hlavním oknem a několika dalšími formuláři. Hlavní okno obsahuje sedm tematicky oddělených záložek, realizovaných pomocí ribbonového menu. Ke každé této záložce se ve vrchní části okna zobrazují příslušné ovládací prvky. V hlavní části okna je pak příslušný obsah. Všechny záložky hlavního okna a některé důležité formuláře jsou dále postupně popsány, stejně tak i související formuláře. Ukázka hlavního okna aplikace je zobrazena na obrázku (Obrázek 4.2).

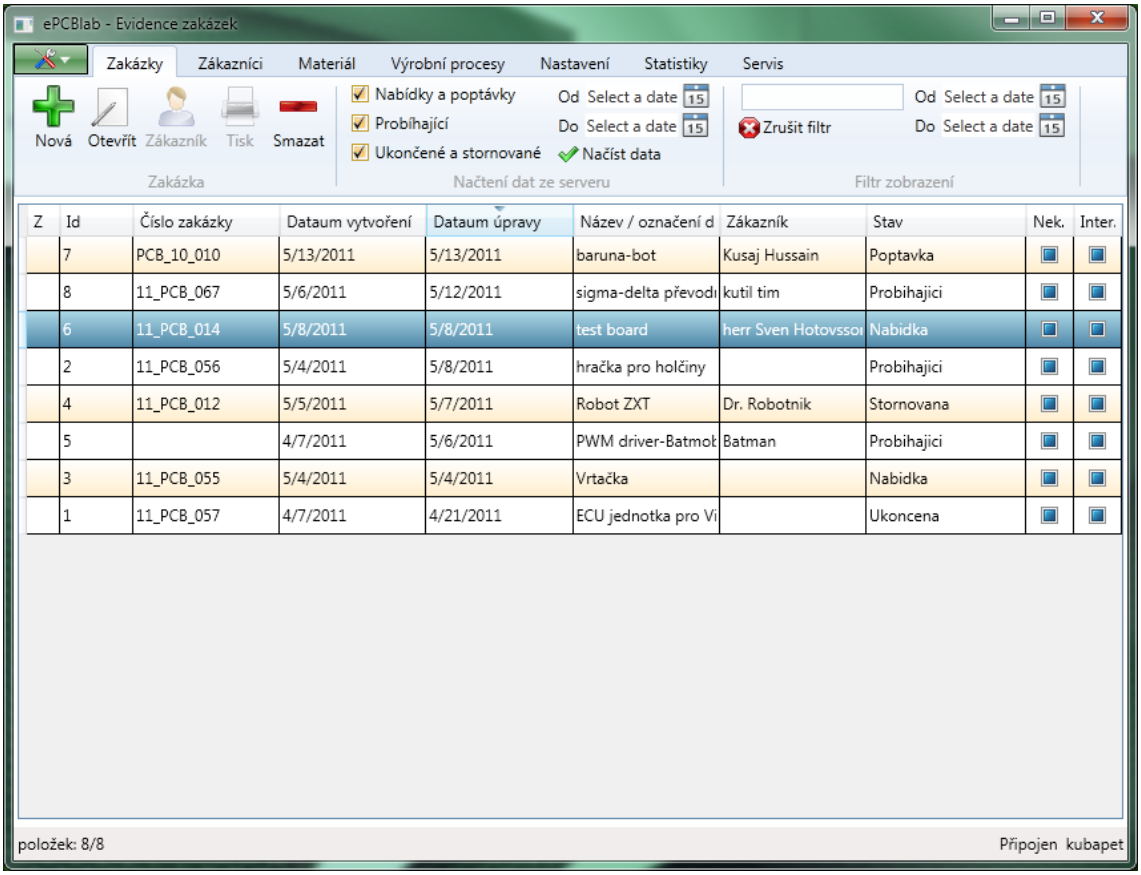
4.2.1 Záložka zakázky

Po spuštění aplikace se otevře hlavní okno na první záložce pojmenované ***Zakázky***. Tato záložka představuje nejdůležitější část aplikace. Zobrazuje přehled zakázek a umožňuje s jednotlivými zakázkami manipulovat. V ribbonovém menu jsou na této záložce tři sekce ovládacích nástrojů. V sekci ***Zakázka*** jsou tlačítka pro přidání nové zakázky, zobrazení detailů zakázky, zobrazení detailů o zákazníkovi, tisk zakázkového dokumentu a smazání zakázky. V sekci ***Načtení dat ze serveru*** je možné vybrat kritéria pro načtení zakázek ze serveru. Pokud bude v evidenci mnoho záznamů, je nesmyslné je explicitně načítat všechny, včetně dávno uzavřených zakázek. Po spuštění aplikace se tedy načtou ze serveru pouze neukončené zakázky. Pomocí tlačítka ***Načíst data*** je zde možné načíst ze serveru zakázky za zvolené období nebo podle jejich

současného stavu. V poslední sekci ***Filtr zobrazení*** je možné načtené zakázky filtrovat podle data nebo podle řetězce, který musí být obsažen v názvu desky nebo v čísle zakázky. Tlačítkem ***Zrušit filtr*** se opět zobrazí všechny načtené položky.

Ve hlavní části okna je samotná tabulka se seznamem zakázek. Zobrazené zakázky je možné třídit vzestupně i sestupně podle jednotlivých sloupců kliknutím na jejich záhlaví. V prvním sloupci zleva se zobrazuje symbol zámečku, pokud je položka momentálně uzamčena. V dalších sloupcích jsou vybrány nejdůležitější atributy zakázky: číslo zakázky, název desky, datum vytvoření, datum poslední úpravy, jméno zákazníka, stav a příznaky zdali se jedná o komerční či interní zakázku.

Pod tabulkou zakázek se nachází stavový řádek, který obsahuje informaci o počtu zobrazených / načtených položek. Dále pokud aplikace načítá položky, se zde zobrazuje „progressbar“ a nakonec v pravé části je indikátor stavu připojení k serveru.



Obrázek 4.2 Hlavní okno se seznamem zakázek

4.2.2 Editační formulář zakázky

Formulář je možné otevřít dvěma způsoby. Prvním z nich je stisknutí tlačítka pro přidání nové zakázky v hlavním okně, v záložce ***Zakázky***. Druhou možností je výběr

konkrétní zakázky ze seznamu na téže záložce a stisknutí tlačítka ***Otevřít***. Formulář se otevírá v samostatném okně na první z jeho záložek, pojmenované ***Základní údaje*** (viz. Obrázek 4.3). Obsahuje celkem pět záložek, členěných podle způsobu užití.

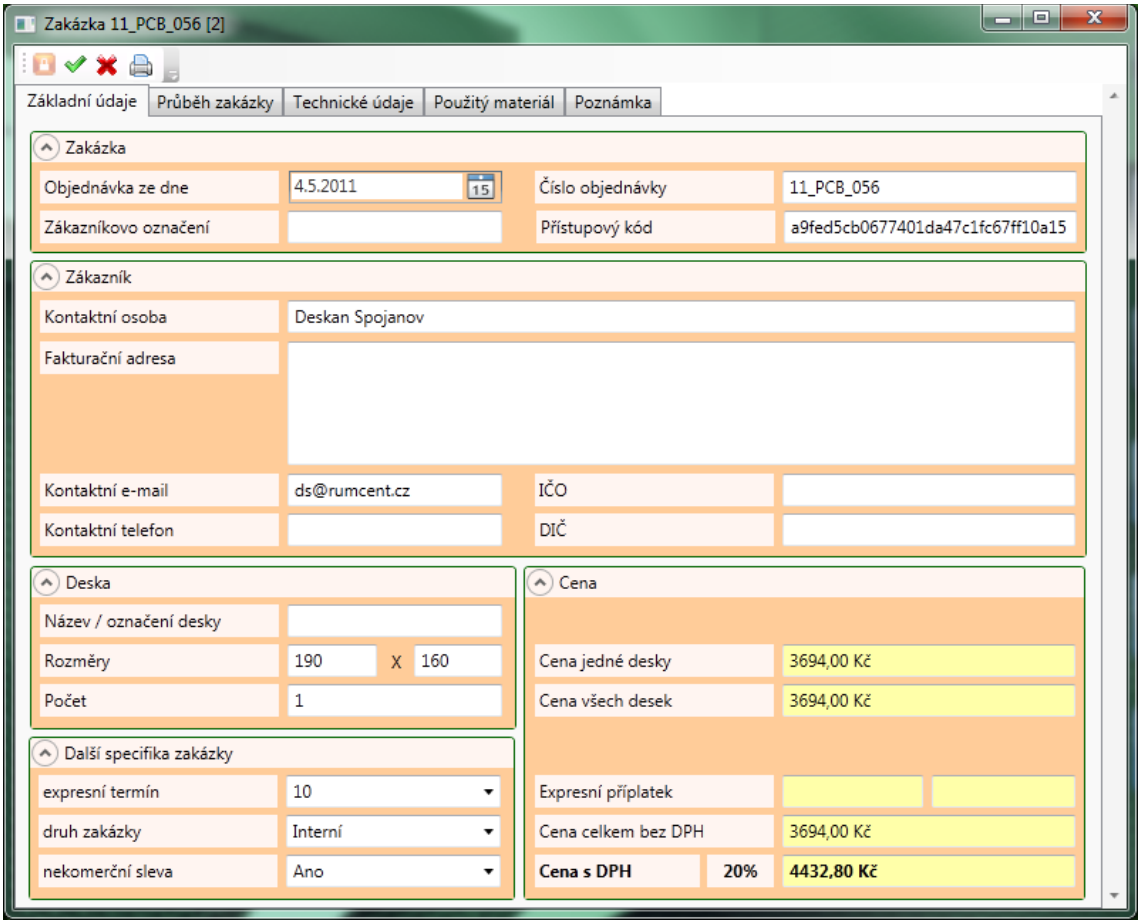
V horní části formuláře je lišta s tlačítky pro uzamčení a uložení dané zakázky, opuštění formuláře a pro tisk aktuální záložky. Pro umožnění editace údajů je třeba zakázku nejprve uzamknout. Uzamčení se provádí tlačítkem ***Uzamknout*** (na tlačítku je symbol zámečku). Tím se zabrání, v daném okamžiku, v přístupu ostatních uživatelů systému k editované zakázce. Tlačítkem ***Uložit*** (symbol zeleného zatržítka) se provedené změny uloží a zakázka se odemkne. Tlačítkem ***Opustit*** (symbol červeného křížku) se formulář uzavře. Pokud byla zakázka před stiskem tlačítka ***Opustit*** uzamčena, aplikace se dotáže, jestli uživatel chce formulář skutečně zavřít a přijít tak o provedené změny. V případě že uživatel souhlasí se ztrátou změn, zakázka se odemkne. Tlačítko ***Tisk*** funguje v závislosti na právě otevřené záložce. Nejčastější využití bude mít pro tisk záložky ***Základní údaje***, ta se využívá jako předávací protokol a podklad pro fakturaci. A dále pro tisk záložky ***Technické informace***, u které se předpokládá využití jako technologická průvodka. Technologická průvodka putuje jako papírový dokument se zakázkou po jednotlivých operacích a slouží k rychlé orientaci o provedení a termínech zakázky.

Záložky ***Základní údaje*** a ***Technologické údaje*** se obsluhují shodně. Pro editaci jsou přístupná pouze bílá políčka a význam dané položky je zřejmý z popisu vlevo od daného políčka. Žlutá políčka jsou pouze pro čtení a jsou v nich vypočtené hodnoty podle rozsahu a konkrétního provedení zakázky (tj. ceny, rozměry, počty kusů apod.).

Záložka ***Průběh zakázky*** se vyplňuje podle skutečně provedených operací. Tím se udržuje přehled o stavu rozpracovanosti zakázky. Tedy například, provede-li pracovník prokovení otvorů, zapíše do formuláře datum, čas a status Hotovo atd. Záložka je tedy jakýmsi seznamem provedených operací.

Záložka ***Materiál*** se generuje automaticky. A je především vstupem pro celkovou evidenci materiálu a výpočet jeho spotřeby a ceny v zakázce.

Smysl záložky ***Poznámka*** je zřejmý.



Obrázek 4.3 Formulář editace zakázky

4.2.3 Záložka zákazníci

Její funkčnost je obdobná jako u záložky ***Zakázky***, ale slouží, jak název napovídá, k evidenci, úpravám a přehledu údajů o zákaznících. Zákazníky zde lze přidávat, editovat a odebírat, případně podle různých kritérií zobrazovat či vybírat. Všechna potřebná tlačítka jsou intuitivní a umístěna opět v horní části v ribbonovém menu.

4.2.4 Editační formulář zákazníka

Tlačítkem ***Zobrazit*** v ribbonovém menu se otevře formulář s přehledně uspořádanými údaji o zákazníkovi. Obsluha formuláře je analogická formuláři zakázek. Před editací je potřeba nejprve provést uzamčení) formuláře (tlačítko ***Uzamknout*** na horní liště, pak teprve je možné údaje editovat. Po uložení či opuštění formuláře se databáze zákazníků opět odemkne a umožní editaci ostatním uživatelům přihlášeným do systému.

4.2.5 Záložky Operace a Materiál

Tyto dvě záložky mají podobnou strukturu. Jsou to nástroje pro zadávání a úpravu vstupních údajů pro výpočty v zakázkách. V ribbonových menu jsou pod jednotlivými tlačítky všechny prováděné operace a veškerý používaný materiál. Po rozkliknutí je možné editovat jednotlivé parametry operací (časy, násobnosti, amortizace…) a informace o materiálech (nákupní ceny, nákupní množství, údaje pro objednávání a především jeho spotřeba). Tyto údaje bude potřeba pečlivě a systematicky naplnit, je to práce pro technologa s potřebnou praxí a dokonale znalého problematiky výroby DPS. Není to tedy předmětem této bakalářské práce.

4.2.6 Záložka Nastavení

Toto je pouze jednoúrovňová tabulka s jednotlivými políčky pro zadávání výchozích hodnot proměnných a konstant vyskytujících se ve výpočtech. Jsou to například: maximální vyrobitelný rozměr technologického panelu, rozměr technologického okolí, délka amortizace strojů, hodinové sazby jednotlivých profesí atd.

4.2.7 Záložka Statistiky

Tato záložka je připravena pro realizaci převážně ekonomických výstupů a analýz. V grafech a tabulkách zde bude možné sledovat různé ukazatele za dané období. Analyzovat a testovat změny různých faktorů jako jsou změny cen materiálů a využívaných služeb, mzdové náklady, množství zakázek pro jednotlivé subjekty atd.

4.2.8 Záložka servis

Poslední záložkou uživatelského rozhraní je servisní záložka, která slouží především k diagnostice a nastavení komunikace s aplikačním serverem. V ribbonovém menu je možné přepínat mezi čtyřmi různými pohledy, které se zobrazují ve vrchní polovině hlavní části okna. Menu je společné pro všechny tyto pohledy, proto v každém pohledu jsou aktivní pouze některé ovládací prvky. Ve druhé polovině okna se nachází konzole, která zaznamenává průběh komunikace se serverem.

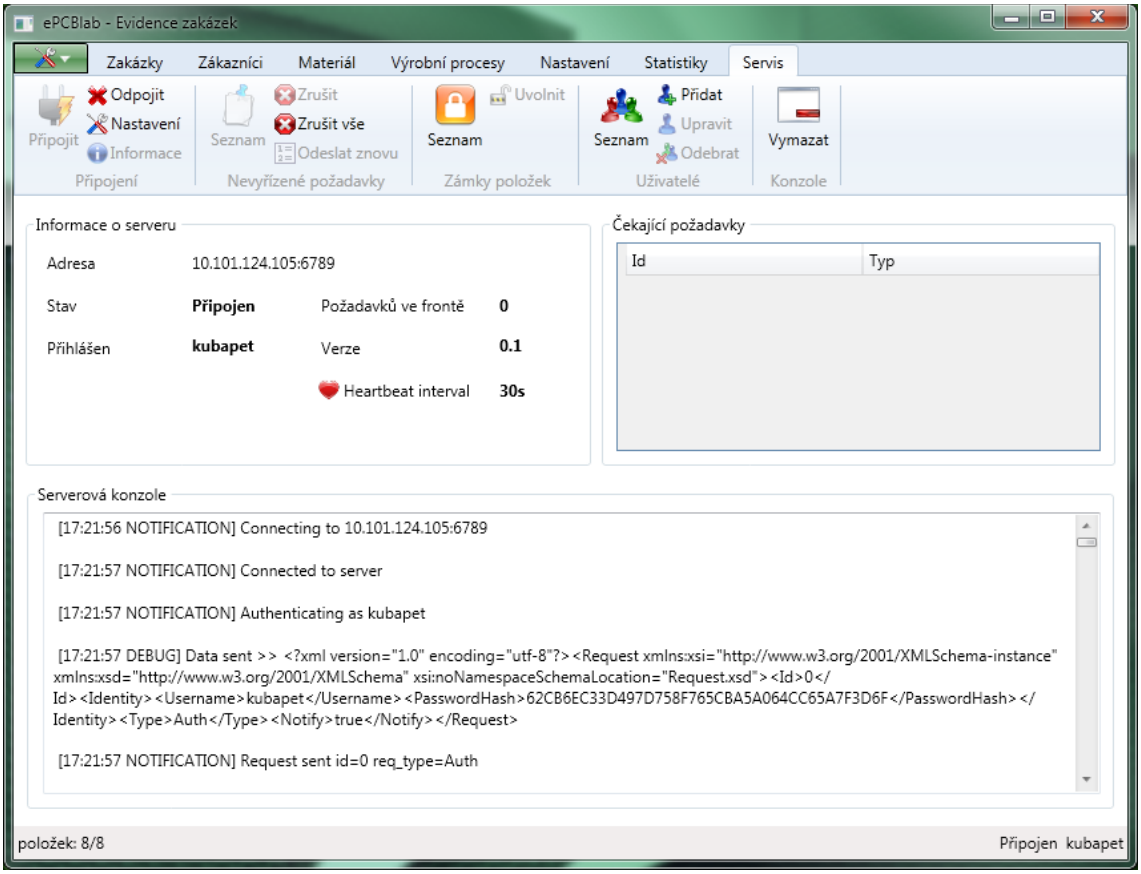
První pohled zobrazuje stav připojení a frontu požadavků. Tento pohled je na této záložce výchozím pohledem po spuštění aplikace. Zobrazit se ale dá kdykoliv stisknutím tlačítka Seznam v sekci ***Nevyřízené požadavky*** nebo tlačítka informace v sekci ***Připojení***. V informacích o stavu připojení je zobrazena adresa serveru, stav

připojeno/odpojeno, jméno přihlášeného uživatele, počet požadavků ve frontě a heartbeat interval. O heartbeat intervalu si můžete přečíst více v kapitole 3.1.4. Nevyřízené požadavky je možné individuálně nebo hromadně stornovat, nebo případně poslat znovu.

Ve druhém pohledu se nachází nastavení připojení. Tento pohled lze otevřít ze sekce ***Připojení*** tlačítkem ***Nastavení***. Zde je možné zadat adresu a port aplikačního serveru, uživatelské jméno a heslo a nastavit zaznamenávání komunikace. K nastavení zaznamenávání slouží dvě zaškrtávací políčka v pravé části. Prvním z nich se dá zaznamenávání komunikace povolit či zakázat. Druhým políčkem se povolí nebo zakáže zaznamenávání detailních zpráv sloužících pro ladění aplikace. Pro běžný provoz toto však není vhodné. Zaznamenávání ladících zpráv může výrazně zpomalit načítání položek ze serveru.

Třetí pohled zobrazuje všechny zámky položek, které se v systému aktuálně nacházejí. Tento pohled je přístupný kliknutím na tlačítko ***Seznam*** v sekci ***Zámky položek***. Po výběru konkrétní položky je možné jí vynuceně odemknout. To je možné ovšem jen v případě, že položku zamkl stejný uživatel, jako ten, který se jí takto pokouší odemknout. V opačném případě je požadavek na odemčení serverem odmítnut. Tato funkce je užitečná pokud se například uživateli náhle během editace položky vypne počítač a relace je násilně ukončena. Zámek v systému zůstane a touto cestou je možné jej uvolnit. Dále tuto funkčnost může uživatel využít, pokud zapomene otevřenou editaci některé položky z relace na jiném počítači. Vynucené odemknutí se provede stiskem tlačítka ***Uvolnit*** se symbolem odemčeného zámečku.

Poslední ze čtyř pohledů v záložce servis slouží k prohlížení, editaci a přidávání uživatelů pro přístup do systému. Pohled se aktivuje stiskem tlačítka ***Seznam*** v sekci ***Uživatelé***. Tato funkcionalita nabude na větším významu teprve s implementací uživatelských rolí, které v současné verzi ePCBLabu implementovány nejsou. Počítá se s nimi však do budoucna.



Obrázek 4.4 Hlavní okno otevřené na servisní záložce

4.3 Systémové požadavky a instalace

Aplikaci není nutné instalovat, celý adresář stačí pouze zkopírovat do libovolného umístění a spustit ePCBLab.exe. Pro správný chod programu je nutné mít v systému nainstalovaný Microsoft .NET Framework 4.0. Aplikace není náročná na systémové prostředky, v paměti i na disku zabere řádově jednotky MB. Nutné je však zajistit síťové spojení s aplikačním serverem. Díky snadné přenositelnosti aplikace (aplikace typu Portable) je možné mít aplikaci na flash disku nebo připravenou na serveru ke stažení, což umožní administrovat ePCBLab po internetu téměř odkudkoliv.

5 Webové klientské rozraní

Prostřednictvím webového rozhraní ePCBLabu mohou zákazníci zadávat poptávky, schvalovat nabídky a sledovat průběh zakázek. Schvalování nabídek je zahrnuto ve stejné sekci jako sledování průběhu zakázky. Tím pádem má celé webové rozhraní pouze dvě hlavní sekce, které jsou přístupné přes rozcestník na úvodní stránce. Tento rozcestník obsahuje dvě tlačítka s odkazy na zmíněné sekce. Obě tato tlačítka by bylo vhodné zakomponovat také do webové prezentace PCB Laboratoře.

5.1 Architektura webového rozhraní a organizace zdrojových kódů

Webové klientské rozhraní je tvořeno aplikací napsanou v programovacím jazyce Java, čímž je zajištěna funkcionalita na různých platformách. Použita byla technologie Java Servlets a Java Server Pages (JSP). Popis využití těchto technologií lze nalézt v publikaci [5]. Celá aplikace vychází ze vzoru softwarové architektury MVC. Je tedy rozdělena na tři části. První z nich je datový model realizovaný obdobným způsobem jako u desktopové aplikace. Druhou částí jsou pohledy. Každému pohledu náleží jeden JSP soubor, který generuje klasický HTML kód s příslušnými formuláři. Poslední částí je kontrolér, který je realizovaný formou jednoho Java Servletu pojmenovaného *Dispatcher*. Komunikace se serverem ePCBLabu se navazuje jak v *Dispatcheru* tak i přímo v jednotlivých pohledech, aby bylo možné ve formulářích zobrazovat získané hodnoty. K vytvoření spojení se serverem z kontextu JSP souboru je využita metoda *UseBean*.

Servlet *Dispatcher* je volán s každým požadavkem na zobrazení webové stránky. Každý takovýto požadavek zhodnotí a na základě tohoto zhodnocení provede příslušnou akci. Pokud je potřeba, pomocí aplikačního klienta odešle požadavek aplikačnímu serveru a vyčká na odpověď. Bližší popis aplikačního klienta se nachází v kapitole 5.1.2. Podle charakteru odpovědi *Dispatcher* dále vybere konkrétní stránku, která se má zobrazit. Pokud není potřeba server kontaktovat (například při úvodním načtení webového rozhraní) *Dispatcher* vybere konkrétní stránku přímo na základě přijatého požadavku na zobrazení webové stránky.

Struktura aplikace webového klientského rozhraní na úrovni komunikační vrstvy (obsahující implementaci protokolu XOE) a na úrovni vrstvy obsahující aplikační

logiku je velmi podobná desktopové klientské aplikaci. V Javě se nepoužívají jmenné prostory jako v případě platformy .NET, místo nich jsou zde však balíčky, které plní odpovídající funkcionalitu. Tato aplikace je kromě JSP souborů s pohledy a balíčku *Controller*, obsahující třídu *Dispatcher*, tvořena dalšími pěti balíčky – *xoe, networking, functions, epcblab* a *datamodel*. Některé z nich jsou v této kapitole popsány blíže. Vzhledem ke zmiňované podobnosti s desktopovou klientskou aplikací je popis napsán převážně formou rozdílů oproti ní.

5.1.1 XOE a Networking

Balíčky *networking* a *xoe* obsahují zjednodušenou implementaci síťového klienta a XOE klienta. Oproti desktopové verzi. zde je vynechána část, která zajišťuje udržování persistentního spojení s aplikačním serverem. Vyřizování požadavků na server a příjem odpovědi proběhne vždy pouze jednorázově při načtení stránky. Jakmile je obdržena odpověď, spojení se ukončí a výsledek ovlivní obsah zobrazené stránky. Tato implementace XOE klienta tedy nepožaduje ani příjem zpráv o událostech v systému (*Notification*). Vynechána je také fronta požadavků. Webové rozhraní čeká vždy na příjem odpovědi na předchozí požadavek, než odešle další požadavek.

5.1.2 EPCBLab

V balíčku epcblab se nachází zjednodušená implementace aplikačního klienta systému ePCBLab a potřebná aplikační logika. Na rozdíl od desktopové klientské aplikace, zde není nutné uchovávat ve vyrovnávací paměti položky načtené ze serveru. Tato implementace poskytuje pouze funkce nezbytné pro webové rozhraní. Mezi tyto funkce patří například načtení zakázky na základě zadaného přístupového kódu, nebo vložení nové zakázky do systému. Aplikační klient webového rozhraní se k serveru může přihlašovat s libovolným uživatelským jménem a heslem. Tyto údaje je však nutné předem nakonfigurovat.

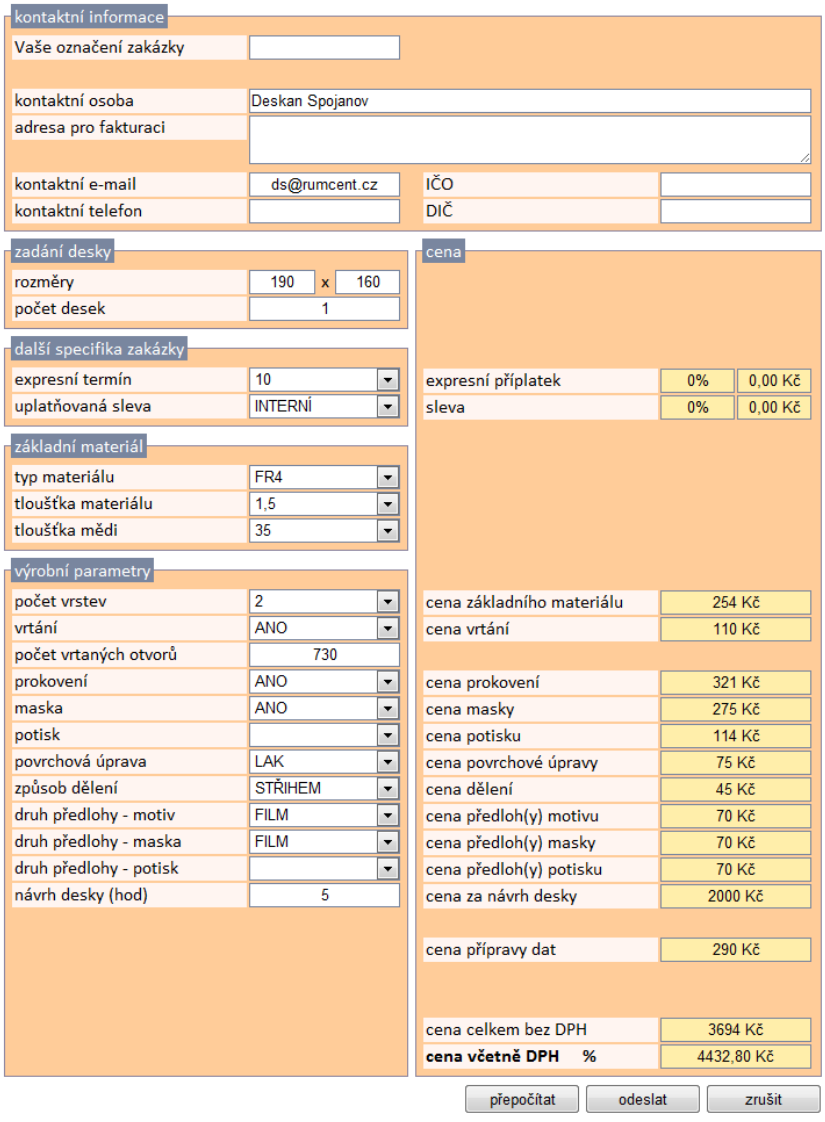
5.2 Popis uživatelského rozhraní

Do prvního kontaktu s aplikací se zákazník dostane na webových stránkách ePCBLabu, kde si může vybrat, jestli zadá novou poptávku nebo bude sledovat již běžící zakázku. K tomuto účelu jsou na stránkách připravena dvě vstupní tlačítka do systému.

5.2.1 Nová poptávka

Tato volba otevře poměrně jednoduchý formulář. I když někomu se může zdát na první pohled složitý, tak protože není potřeba vyplňovat všechny údaje, je odeslání poptávky velmi jednoduché. Stačí vyplnit pouze jméno, e-mailovou adresu, rozměr desky, počet kusů a počet vrstev a poptávku odeslat (tlačítko ***Odeslat*** dole pod formulářem). Nicméně zákazník si může, zadáváním ostatních parametrů, také otestovat jaký vliv na cenu má například vynechání některé operace, změna požadovaného počtu kusů a tak dále. Přepočítání lze provést příslušným tlačítkem dole (viz. Obrázek 5.1).

S odesláním první poptávky je vygenerován přístupový kód a odeslán zákazníkovi na uvedený mail. Poptávka je pracovníky PCBlabu zpracována doplněna o potřebné údaje a uložena jako nabídka.



Obrázek 5.1 Webový formulář pro zadání nové poptávky

5.2.2 Sledování zakázek

Pomocí obdrženého kódu může zákazník sledovat průběh zakázky. Touto volbou je zákazník nejprve vyzván k zadání přístupového kódu a poté se otvírá identický formulář s upřesněnými údaji pracovníkem PCBlabu a vypočtenou výslednou cenou. Zákazník může nabídku odsouhlasit (tlačítko ***Potvrdit*** dole pod formulářem) nebo ještě změnit některé parametry jako je počet kusů, vybrané operace atd. a nechat nabídku znovu přepočítat. Po odsouhlasení je zasláno na e-mail potvrzení o přijetí a zakázka se začne zpracovávat.

I v průběhu zpracování může zákazník sledovat aktuální průběh své zakázky. Pomocí odkazu Sledování zakázek a zadání přístupového kódu si může ve formuláři prohlédnout, které operace jsou již hotové. Ve formuláři nejde v takový okamžik již nic měnit, ale hotové operace jsou zvýrazněny. Jsou zvýrazňovány na základě údajů zadávaných pracovníky PCBlabu v desktopovém klientu systému ePCBLab. Po označení všech operací jako provedené je mimo jiné odesláno na e-mailovou adresu oznámení o hotové zakázce připravené k odběru.

5.3 Systémové požadavky a instalace

Webové rozhraní ePCBLabu ke svému provozu potřebuje webový server s podporou JSP a Java servetů. Doporučeným a úspěšně otestovaným webovým serverem je Tomcat6 nebo Tomcat7 od společnosti Apache, který je dostupný jako open source. Webové rozhraní není příliš náročné ani na výkon ani další systémové zdroje. Na pevném disku zabere necelý 1MB a v operační paměti maximálně několik jednotek MB. Je však nutné zajistit kvalitní síťové spojení s aplikačním serverem ePCBLabu. Samozřejmě je možné provozovat webové rozhraní na stejném serveru jako aplikační server ePCBLabu.

Instalace webového rozhraní není příliš složitá, ale vyžaduje znalost konfigurace webového serveru. Existuje několik možností jak webové rozhraní nainstalovat. Vhodným postupem může být například postup uvedený v následujícím odstavci. Tento postup byl použit i v případě vývojového serveru, na kterém bylo webové rozhraní testováno. Vývojový server disponoval operačním systémem OpenSUSE, čemuž je následující popis uzpůsoben. Zkušenější systémový administrátor však bez problémů zvládne pomocí tohoto popisu webové rozhraní zprovoznit i na jiných systémech.

Nejprve je nutné na vhodné místo na serveru zkopírovat adresář s webovým rozhraním. Následně nainstalovat pomocí správce balíčku YaST webový server Tomcat v aktuální verzi, se všemi potřebný závislostmi. V současné stabilní verzi OpenSUSE je k dispozici Tomcat6. Poté je třeba přidat virtuálního webového hostitele do konfiguračního souboru webového serveru Tomcat, pojmenovaného *server.xml*. Tento soubor se obvykle nachází v adresáři */etc/tomcat6/*. Přidání virtuálního webového hostitele lze provést například vložením následujícího kódu do sekce Server – Service – Engine ve zmíněném souboru.

```
<Host name="epcblab.example.org">
  <Context path="" docBase="/srv/www/pages/orgexample.epcblab" />
</Host>
```

Uvedené jméno hostitele a cestu k webovému adresáři je samozřejmě nutné podle dané situace upravit. Nakonec je potřeba webový server Tomcat spustit příkazem *service tomcat6 start*. Pokud je vše správně nastaveno, po zadání adresy http://epcblab.example.org:8080 do webového prohlížeče by se mělo zobrazit webové rozhraní ePCBLabu.

Port 8080 je výchozím portem, na kterém server Tomcat naslouchá. Port lze samozřejmě změnit v konfiguračním souboru *server.xml* například na port 80, který je výchozím portem protokolu http. Problém může nastat, pokud již na daném serveru běží jiný webový server. V případě Apache je ale možné použít modul *mod_jk*, který dokáže propojit Apache s Tomcatem. Ačkoliv byl na vývojovém serveru tento modul použit, popis jeho zprovoznění je nad rámec této práce.

Závěr

Výsledkem této práce jsou dvě klientské aplikace systému ePCBLab s uživatelským rozhraním. Tyto aplikace jsou postaveny na propracovaném jádře, které je nyní možné snadno rozšiřovat o další funkcionalitu. Systém ePCBLab je velmi komplexní evidenční systém, který by v komerční sféře jistě realizoval větší tým lidí. V příloze Příloha B – seznam použitých technologií je uveden seznam použitých technologií a nástrojů. V ideálním případě by systém byl vyvíjen týmem, který by disponoval jedním odborníkem pro každou z uvedených technologií.

Celý systém by bylo vhodné v dohledné době dále rozšířit. Přínosná by byla zejména podpora uživatelských rolí, se kterou je v systému počítáno. Na straně serveru by byl definován seznam uživatelských rolí, kde každá z nich by obsahovala seznam povolených typů operací nad různými datovými entitami. Každému uživateli by pak byla přiřazena konkrétní role. Toto rozšíření by znamenalo zásah pouze do datového modelu ve všech částech systému a do vrstvy uživatelského rozhraní v klientských aplikacích.

Navržená uživatelská rozhraní jsou z pohledu uživatelů přívětivá, jednoduchá a přehledná. Nicméně i zde je možné implementovat řadu vylepšení, jakým je například našeptávání při vyplňování formulářů. V případě klientské desktopové aplikace by tato úprava nebyla příliš náročná. Oproti tomu klientskou webovou aplikaci by bylo nutné rozšířit značně rozšířit, aby bylo možné našeptávání realizovat. Webové stránky v této aplikaci by museli obsahovat JavaScript a využívat technologii AJAX (Asynchronní Javascript a XML), aby bylo možné zasílat požadavky z prohlížečů na webový server bez nutnosti obnovení stránky.

Jednotlivé aplikace systému jsou snadno přenositelné, aplikační server ePCBLabu i klientské webové rozhraní je možné zprovoznit na libovolném serveru. Obě tyto části je možné libovolně přemístit i za provozu systému. Při volbě správného postupu dojde pouze ke krátkému výpadku. Klientská desktopová aplikace nevyžaduje instalaci, tudíž je možné ji nahrát například na flash disk.

Použitá literatura

[1] NAIK, Dilip C. . *Internet - standardy a protokoly*. Praha,Computer Press, 1999. 303 s. ISBN 80-7226-146-0.

[2] Extensible Markup Language (XML) 1.0 (Fourth Edition) [Online]. World Wide Web Consortium (W3C). 29. 9 2006. URL: <http://www.w3.org/TR/2006/REC-xml-20060816/#charsets>.

[4] PETZOLD, Charles. *Mistrovství ve Windows Prezentation Foundation*. Brno, Computer Press, 2008. 928 s. ISBN 978-80-251-2141-2.

[3] GRIFFITHS, Ian; ADAMS, Matthew; LIBERTY, Jesse. *Programming C# 4.0*. 6th edition. Sebastopol : O'Reilly Media, 2010. 832 s. ISBN 978-0-596-15983-2.

[5] HALL, Marty; BROWN, Larry. *Core Servlets and Java Server Pages* [online]. Sun Microsystems Press, 2011 [cit. 5. května 2011]. URL: <http://pdf.coreservlets.com/>.

Příloha A – diagram tříd datového modelu

Příloha B – seznam použitých technologií

V této příloze jsou popsány veškeré technologie a nástroje použité při realizaci ePCBLau, včetně těch, které byly použité v serverové části.

Programovací, skriptovací a popisovací jazyky

- **XML a XSD** – Pomocí XML jsou formátovány zprávy, pomocí kterých se přenášejí data mezi serverem a klienty. XSD Šablonami je pak pevně dána struktura těchto zpráv.
- **C#** – V programovacím jazyce C# byl napsána větší část klientské desktopové aplikace.
- **XAML** – Značkovací jazyk XAML byl použit pro popis uživatelského rozhraní klientské desktopové aplikace.
- **Java** – V programovacím jazyce Java je vytvořen aplikační server ePCBLabu a dále aplikační a komunikační logika webové klientské aplikace.
- **JSP** – Pomocí JSP jsou generovány jednotlivé stránky webového klientského rozhraní.
- **HTML a CSS** – Pomocí HTML a kaskádových stylů je definována podoba uživatelského rozhraní webové klientské aplikace.
- **Bash** – Linuxový příkazový shell Bash byl použit při instalaci a konfiguraci ePCBLabu.

Protokoly

- **TCP** – Protokol TCP slouží jako transportní vrstva pro přenos zpráv XOE protokolu v systému ePCBLab.
- **SSL** – Pomocí SSL je v ePCBLabu zabezpečen přenos XOE zpráv.

Knihovny a frameworky

- **Microsoft .NET Framework 4.0** – .NET Framework je základem klientské desktopové aplikace.
- **LINQtoXML** – Tato knihovna je součástí .NET Frameworku, v klientské desktopové aplikaci plní úlohu vytváření a čtení XOE zpráv.

- **Windows Presentation Foundation** (WPF) – WPF je součástí .NET Frameworku, jedná se o rozsáhlou knihovnu pomocí které bylo vytvořeno rozhraní klientské desktopové aplikace. Použití WPF je spjato se značkovacím jazykem XAML.
- **Ribbon for WPF** – Rozšíření pro WPF, umožňuje tvorbu ribbonových menu, které jsou známé například z Microsoft Office. Ribbonové menu je rovněž použito v klientské desktopové aplikaci.
- **JavaEE 6** – Rozsáhlý framework programovacího jazyka Java. Obsahuje například Java Servlet API, které bylo použito při tvorbě klientské webové aplikace.
- **Hibernate Framework** – Tento framework byl použit v serverové části ePCBLabu pro práci s databází MySQL.
- **JAXB** – Knihovna pro programovací jazyk Java, v ePCBLabu zajišťuje tvorbu a zpracování XOE zpráv.

Nástroje a podpůrné aplikace

- **Microsoft Visual Studio 2010** – Ve vývojovém prostředí Visual Studio 2010 byla vytvořena klientské desktopová aplikace.
- **Netbeans** – Vývojové prostředí Netbeans umožnilo vývoj aplikačního serveru ePCBLabu a vývoj klientské webové aplikace.
- **Wireshark network analyzer** – Pomocí Wiresharku bylo možné monitorovat provoz na síti a ladit XOE protokol.
- **OpenSUSE Linux** – Distribuce Linuxu OpenSUSE byla použita na testovacím serveru pro systém ePCBLab.
- **Apache Tomcat 6** – Webový server Tomcat byl využit k provozu klientské webové aplikace.
- **MySQL databáze** – Všechny evidované položky v systému ePCBLab se ukládají do databáze MySQL.

Příloha C – obsah přiloženého CD

Data na přiloženém kompaktním disku obsahují zdrojové kódy obou klientských aplikací, sestavené aplikace včetně aplikačního serveru ePCBLabu a text této zprávy ve formátech DOCX a PDF. Organizace dat na CD odpovídá adresářové struktuře znázorněné odrážkami.

- **bin**
  - **desktop** – klientská desktopová aplikace systému ePCBLab
  - **web** – klientská webová aplikace systému ePCBLab
  - **server** – aplikační server ePCBLabu
- **doc**
  - **zprava** – vlastní text bakalářské práce (formáty PDF a DOCX)
  - **podklady** – různý podkladový materiál k této práci, obrázky a diagramy
- **src**
  - **desktop** – zdrojové kódy klientské desktopové aplikace
  - **web** – zdrojové kódy klientské webové aplikace